



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/720,963	11/24/2003	Richard D. Dettinger	ROC920030278US1	5212

46797 7590 03/06/2007

IBM CORPORATION, INTELLECTUAL PROPERTY LAW  
DEPT 917, BLDG. 006-1  
3605 HIGHWAY 52 NORTH  
ROCHESTER, MN 55901-7829

EXAMINER
----------

DWIVEDI, MAHESH H

ART UNIT	PAPER NUMBER
----------	--------------

2168

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	03/06/2007	PAPER

**Please find below and/or attached an Office communication concerning this application or proceeding.**

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

**Office Action Summary**

Application No.

10/720,963

Applicant(s)

DETTINGER ET AL.

Examiner

Mahesh H. Dwivedi

Art Unit

2168

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

- 1) ☒ Responsive to communication(s) filed on 10 August 2006.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

- 4) ☒ Claim(s) 1-26 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☒ Claim(s) 1-26 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

**Application Papers**

- 9) ☒ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 10 August 2006 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.  
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).  
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some \* c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
  - ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
  - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

\* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☒ Information Disclosure Statement(s) (PTO/SB/08)  
Paper No(s)/Mail Date 10/27/2004.
- 4) ☐ Interview Summary (PTO-413)  
Paper No(s)/Mail Date. \_\_\_\_\_
- 5) ☐ Notice of Informal Patent Application
- 6) ☐ Other: \_\_\_\_\_

## **DETAILED ACTION**

### ***Information Disclosure Statement***

1. The information disclosure statements (IDS) submitted on 11/24/2003 and 10/27/2004 have been received, entered into the record, and considered. The submission is in compliance with the provisions of 37 CFR 1.97. Accordingly, the information disclosure statements are being considered by the examiner.

### ***Specification***

2. The disclosure is objected to because of the following informalities:  
Attorney Docket Number at paragraph 1 should be replaced with the Application serial number and its current status

### ***Claim Rejections - 35 USC § 101***

3. 35 U.S.C. 101 reads as follows:

Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

4. Claims 15-19 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. Claims 62-68 appear to represent nonfunctional descriptive material. Descriptive material can be characterized as either "functional descriptive material" or "nonfunctional descriptive material." In this context, "functional descriptive material" consists of data structures and computer programs which impart functionality when employed as a computer component. (The definition of "data structure" is "a physical or logical relationship among data elements, designed to support specific data manipulation functions." The New IEEE Standard Dictionary of Electrical and Electronics Terms 308 (5th ed. 1993).) "Nonfunctional descriptive material" includes but is not limited to music, literary works and a compilation or mere arrangement of data. When nonfunctional descriptive material is recorded on some computer-readable medium, in a computer or on an electromagnetic carrier signal, it is not statutory since no requisite functionality is present to satisfy the practical application requirement. Merely claiming nonfunctional descriptive material, i.e., abstract ideas, stored in a computer-readable medium, in a computer, on an electromagnetic carrier

signal does not make it statutory. See Diehr, 450 U.S. at 185-86, 209 USPQ at 8 (noting that the claims for an algorithm in Benson were unpatentable as abstract ideas because "[t]he sole practical application of the algorithm was in connection with the programming of a general purpose computer."). Such a result would exalt form over substance. See also In re Johnson, 589 F.2d 1070, 1077, 200 USPQ 199, 206 (CCPA 1978) ("form of the claim is often an exercise in drafting"). Thus, nonstatutory music is not a computer component and it does not become statutory by merely recording it on a compact disk. Protection for this type of work is provided under the copyright law.

Claims 15-19 are further rejected under 35 U.S.C 101 because the claimed invention is directed to the non-statutory subject area of electro-magnetic signals, carrier waves. Claims 15-19 recite the limitation "**computer-readable medium containing a program**". The examiner interprets "computer-readable medium containing a program" as a computer-readable medium containing a program defined by the characteristics in paragraph 28 of the applicant's specification. According to paragraph 28 of the applicant's specification, a computer-readable medium containing a program consists of "The program(s) of the product defines functions of the embodiments (including the methods described herein) and can be contained on a variety of **signal-bearing media**. Illustrated **signal-bearing media** include...information conveyed to a computer by a communications medium, such as through a computer or telephone network, including wireless communications. The latter embodiment specifically includes information downloaded from the Internet and other networks. Such **signal-bearing media, when carrying computer-readable instructions** that direct the functions of the present invention, represent embodiments of the present invention". Claims 15-19 recite nothing but the physical characteristics of a form of energy, such as a frequency, voltage, or the strength of a magnetic field, define energy or magnetism, per se, and as such are nonstatutory natural phenomena. O'Reilly, 56 U.S. (15 How.) at 112-14. Moreover, a claim reciting a signal encoded with functional descriptive material does not fall within any of the categories of patentable subject matter set forth in § 101. First, a claimed signal is clearly not a "process" under § 101 because it is not a series of steps. The other three § 101 classes of machine,

Art Unit: 2168

compositions of matter and manufactures "relate to structural entities and can be grouped as 'product' claims in order to contrast them with process claims." 1 D. Chisum, Patents § 1.02 (1994). The three product classes have traditionally required physical structure or material. "The term machine includes every mechanical device or combination of mechanical device or combination of mechanical powers and devices to perform some function and produce a certain effect or result." *Corning v. Burden*, 56 U.S. (15 How.) 252, 267 (1854). A modern definition of machine would no doubt include electronic devices which perform functions. Indeed, devices such as flip-flops and computers are referred to in computer science as sequential machines. A claimed signal has no physical structure, does not itself perform any useful, concrete and tangible result and, thus, does not fit within the definition of a machine. A "composition of matter" "covers all compositions of two or more substances and includes all composite articles, whether they be results of chemical union, or of mechanical mixture, or whether they be gases, fluids, powders or solids." *Shell Development Co. v. Watson*, 149 F. Supp. 279, 280, 113 USPQ 265, 266 (D.D.C. 1957), *aff'd*, 252 F.2d 861, 116 USPQ 428 (D.C. Cir. 1958). A claimed signal is not matter, but a form of energy, and therefore is not a composition of matter. The Supreme Court has read the term "manufacture" in accordance with its dictionary definition to mean "the production of articles for use from raw or prepared materials by giving to these materials new forms, qualities, properties, or combinations, whether by hand-labor or by machinery." *Diamond v. Chakrabarty*, 447 U.S. 303, 308, 206 USPQ 193, 196-97 (1980) (quoting *American Fruit Growers, Inc. v. Brogdex Co.*, 283 U.S. 1, 11, 8 USPQ 131, 133 (1931), which, in turn, quotes the Century Dictionary). Other courts have applied similar definitions. See *American Disappearing Bed Co. v. Arnaelsteen*, 182 F. 324, 325 (9th Cir. 1910), *cert. denied*, 220 U.S. 622 (1911). These definitions require physical substance, which a claimed signal does not have. Congress can be presumed to be aware of an administrative or judicial interpretation of a statute and to adopt that interpretation when it re-enacts a statute without change. *Lorillard v. Pons*, 434 U.S. 575, 580 (1978). Thus, Congress must be presumed to have been aware of the interpretation of manufacture in *American Fruit Growers* when it passed the 1952 Patent

Art Unit: 2168

Act. A manufacture is also defined as the residual class of product. 1 Chisum, § 1.02[3] (citing W. Robinson, The Law of Patents for Useful Inventions 270 (1890)). A product is a tangible physical article or object, some form of matter, which a signal is not. That the other two product classes, machine and composition of matter, require physical matter is evidence that a manufacture was also intended to require physical matter. A signal, a form of energy, does not fall within either of the two definitions of manufacture. Thus, a signal does not fall within one of the four statutory classes of § 101.

To expedite a complete examination of the instant application, the claims rejected under 35 U.S.C. 101 (nonstatutory) above are further rejected as set forth below in anticipation of applicant amending these claims to place them within the four categories of invention.

***Claim Rejections - 35 USC § 102***

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

6. Claims 1-20, and 22-26 are rejected under 35 U.S.C. 102(a) as being anticipated by **Young** (U.S. Patent 6,560,606).

7. Regarding claim 1, **Young** teaches a method comprising:

A) providing an interface for specifying a plurality of functional modules (Column 2, lines 58-67-Column 3, lines 1-4);

B) providing a configuration file containing information regarding invocation of the functional modules (Column 2, lines 52-57, Column 9, lines 12-16, lines 54-63); and

C) invoking the plurality of functional modules in a manner determined according to information retrieved from the configuration file (Column 9, lines 13-33).

The examiner notes that **Young** teaches “**providing an interface for specifying a plurality of functional modules**” as “The processing modules, referred to herein also as “plug-ins”, can operate under the control of an execution management framework. The plug-ins can be viewed as plugging into and out of the framework, and

Art Unit: 2168

as modular, reusable computational pieces or building blocks, which come together to perform the computation under the direction of the framework and pursuant to a configuration file. The plug-ins can be added, removed, and/or replaced for modifying the calculation performed by the system in generating output data. For use, a user or operator devices the computation, e.g., calculation formula, required for a particular VAS, and uses the configuration manager's user interface to select and order the plug-ins to carry out that formula" (Column 2, lines 58-67-Column 3, lines 1-4). The examiner further notes that **Young** teaches **"providing a configuration file containing information regarding invocation of the functional modules"** as "Each processing module performs a specific sub-part of a computation on the metered information, and the configuration manager generates a configuration file for specifying the order of operation, computation sub-part, and other operational parameters for the individual processing modules" (Column 2, lines 52-57) and "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24). The examiner further notes that **Young** teaches **"invoking the plurality of functional modules in a manner determined according to information retrieved from the configuration file"** as "As illustrated, those of the plug-ins nos. 1-8 that can be processed in parallel are shown in a vertical stack in the drawing (as in the case, e.g., of plug-ins 1, 2 and 3; or plug-ins 6 and 7; or plug-ins 8 and 9). Moreover, those of the plug-ins nos. 1-8 that are dependent on, and therefore need to be processed after, other plug-ins are shown to the right of the other plug-ins on which they depend (as in the case, e.g., of plug-in 5

Art Unit: 2168

dependent on plug-in 4 and thus shown in the drawing, to the right of plug-in 4)" (Column 9, lines 24-33).

Regarding claim 2, **Young** further teaches a method comprising:

A) wherein the interface is a graphic user interface utilized by users to specify functional modules (Column 2, lines 58-67-Column 3, lines 1-4, lines 59-63).

The examiner notes that **Young** teaches "**wherein the interface is a graphic user interface utilized by users to specify functional modules**" as "The processing modules, referred to herein also as "plug-ins", can operate under the control of an execution management framework. The plug-ins can be viewed as plugging into and out of the framework, and as modular, reusable computational pieces or building blocks, which come together to perform the computation under the direction of the framework and pursuant to a configuration file. The plug-ins can be added, removed, and/or replaced for modifying the calculation performed by the system in generating output data. For use, a user or operator devices the computation, e.g., calculation formula, required for a particular VAS, and uses the configuration manager's user interface to select and order the plug-ins to carry out that formula" (Column 2, lines 58-67-Column 3, lines 1-4) and "The system permits NSPs and ISPs to create new rating schemes and cross-service plans by entering the appropriate computational parameters using the configuration manager's user interface, e.g., a graphical user interface (GUI)" (Column 3, lines 59-63).

Regarding claim 3, **Young** further teaches a method comprising:

A) wherein the allows an external application to specify functional modules (8, lines 19-33).

The examiner notes that **Young** teaches "**wherein the allows an external application to specify functional modules**" as "The pipeline controller 210 can be a computer-executed program stored and executed, for example, on one of the machines 202, 204, 206 or a separate machine (not shown). The stages 202, 204, 206 are chained (i.e., connected) together via message queues 212, 214 connected in the data



Art Unit: 2168

stream between the adjacent stages for passing session objects therebetween, including partially or wholly processed objects. The queues 212, 214 enable non-synchronous operation of the stages. Each of the stages 202, 204, 206 coordinates the execution of a number of processing modules ("plug-ins") 220, supported by an execution management framework 225. The plug-ins 220 can be viewed as plugging into and out of the execution management framework 225, depending on computational requirements" (Column 8, lines 19-33).

Regarding claim 4, **Young** further teaches a method comprising:

- A) wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the plurality of functional modules (Column 3, lines 32-48, lines 59-63); and
- B) the configuration file contains information relating the plurality of functional modules to the multi-analysis functional modules (Column 3, lines 32-48, Column 13, lines 59-67-Column 14, lines 1-6, Figure 6F)

The examiner notes that **Young** teaches "**wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the plurality of functional modules**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of

Art Unit: 2168

functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6). The examiner further notes that **Young** teaches **"the configuration file contains information relating the plurality of functional modules to the multi-analysis functional modules"** as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues

Art Unit: 2168

when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6).

Regarding claim 5, **Young** further teaches a method comprising:

A) wherein the configuration file contains an explicit sequence in which the functional modules should be executed (Column 2, lines 52-57, Column 3, lines 32-48, Column 9, lines 12-16, lines 54-63)

The examiner notes that **Young** teaches "**wherein the interface is a graphic user interface utilized by users to specify functional modules**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48):

Regarding claim 6, **Young** further teaches a method comprising:

A) wherein the configuration file contains information indicating one or more parameters required for invoking each of the functional modules (Column 9, lines 12-16, lines 54-63, Column 14, lines 46-60); and

Art Unit: 2168

B) invoking the plurality of functional modules comprises invoking only those functional modules whose one or more required parameters are available (Column 9, lines 12-16, lines 54-63, Column 14, lines 46-60).

The examiner notes that **Young** teaches “**wherein the configuration file contains information indicating one or more parameters required for invoking each of the functional modules**” as “A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins” (Column 9, lines 12-24) and “The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed” (Column 14, lines 46-60). The examiner further notes that **Young** teaches “**invoking the plurality of functional modules comprises invoking only those functional modules whose one or more required parameters are available**” as “A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process

Art Unit: 2168

space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

Regarding claim 7, **Young** further teaches a method comprising:

A) wherein invoking the plurality of functional modules in a manner determined according to information retrieved from the configuration file comprises invoking at least two functional modules in parallel (Column 3, lines 22-31, Column 9, lines 12-16, lines 54-63)

The examiner notes that **Young** teaches "**wherein the interface is a graphic user interface utilized by users to specify functional modules**" as "The system can be implemented to include a multi-stage processing pipeline, each pipeline stage including multiple processing modules and an execution management framework, and capable of multi-threading operation to cause the plug-ins to execute in series or in parallel to speed processing by the plug-ins while accommodating computational dependencies. In some embodiments, the system can also have a metering apparatus

Art Unit: 2168

for collecting the metered user information, and a presentation manager for providing processed session data to data consumers" (Column 3, lines 22-31) and "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24).

Regarding claim 8, **Young** further teaches a method comprising:

A) wherein the configuration file is in an extensible markup language (XML) format (Column 10, lines 4-7)

The examiner notes that **Young** teaches "**wherein the interface is a graphic user interface utilized by users to specify functional modules**" as "The configuration manager 150 generates a configuration file for each stage of the pipeline 518, preferably specifying configuration data in XML format" (Column 10, lines 4-7).

Regarding claim 9, **Young** further teaches a method comprising:

A) wherein at least one of the functional modules is a plug-in component of the application (Column 2, lines 58-64)

The examiner notes that **Young** teaches "**wherein at least one of the functional modules is a plug-in component of the application**" as "The processing modules, referred to herein also as "plug-ins", can operate under the control of an execution management framework. The plug-ins can be viewed as plugging into and out of the framework, and as modular, reusable computational pieces or building blocks, which come together to perform the computation under the direction of the framework and pursuant to a configuration file" (Column 2, lines 58-64).

Regarding claim 10, **Young** further teaches a method comprising:

A) wherein at least one of the functional modules is an external application (Column 2, lines 48-67).

The examiner notes that **Young** teaches “**wherein at least one of the functional modules is a plug-in component of the application**” as “The invention resides in a metering and processing system for processing metered information, which incorporates configurable processing modules and a configuration manager. The system can be readily and flexibly configured, responsive to operator directions, to process information to meet the needs of data consumers, such as NSPs and ISPs. Each processing module performs a specific sub-part of a computation on the metered information, and the configuration manager generates a configuration file for specifying the order of operation, computation sub-part, and other operational parameters for the individual processing modules” (Column 2, lines 46-57) and “The processing modules, referred to herein also as “plug-ins”, can operate under the control of an execution management framework. The plug-ins can be viewed as plugging into and out of the framework, and as modular, reusable computational pieces or building blocks, which come together to perform the computation under the direction of the framework and pursuant to a configuration file” (Column 2, lines 58-64).

Regarding claim 11, **Young** teaches a method comprising:

A) obtaining a set of one or more parameters required for invoking the specified functional modules (Column 10, lines 15-35);

B) invoking one or more of the specified functional modules whose required parameters are available in a result set collection (Column 9, lines 13-33, Column 14, lines 55-60);

C) obtaining a result set in response to invoking the one or more functional modules (Column 14, lines 55-60, Column 15, lines 21-35);

D) adding the result set to the result set collection (Column 14, lines 55-60, Column 15, lines 21-35); and

Art Unit: 2168

E) repeating steps (A)-(D) until all the specified functional modules have been executed (Column 15, lines 3-20).

The examiner notes that **Young** teaches “**obtaining a set of one or more parameters required for invoking the specified functional modules**” as “The configuration files configure the stages and plug-ins at three-levels. To configure a pipeline, an operator first selects and loads a stage layout, then selects and loads a layout of the plug-ins within each stage, and then selects and loads individual plug-in parameters, all as will described next” (Column 10, lines 15-20) and “This configuration includes the following configurable parameters: stages on each machine, and arrangement and dependencies between stages”(Column 10, lines 22-24). The examiner further notes that **Young** teaches “**invoking one or more of the specified functional modules whose required parameters are available in a result set collection**” as “As illustrated, those of the plug-ins nos. 1-8 that can be processed in parallel are shown in a vertical stack in the drawing (as in the case, e.g., of plug-ins 1, 2 and 3; or plug-ins 6 and 7; or plug-ins 8 and 9). Moreover, those of the plug-ins nos. 1-8 that are dependent on, and therefore need to be processed after, other plug-ins are shown to the right of the other plug-ins on which they depend (as in the case, e.g., of plug-in 5 dependent on plug-in 4 and thus shown in the drawing, to the right of plug-in 4)” (Column 9, lines 24-33) and “Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed”(Column 14, lines 55-60). The examiner further notes that **Young** teaches “**obtaining a result set in response to invoking the one or more functional modules**” as “Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed”(Column 14, lines 55-60) and “Returning to the example of FIG. 6A, plug-ins C and D can be executed immediately because they do not depend on other plug-ins, as indicated by the zero in the designations C:0 and D:0. Afterwards, as is the case each



Art Unit: 2168

time a plug-in is executed, the dependency counts of counters 670A, . . . for all plug-ins that depend on the executed plug-ins are decremented by decrementer 672.

Accordingly, as shown in FIG. 6C, when plug-in C is executed, the counter for plug-in B (which depends on plug-in C) will be decremented from a value of 2 to a value of 1.

Plug-in B now has only one dependency, plug-in D. As shown in FIG. 6D, when plug-in D is executed, the counter for plug-in B will again be decremented, this time from a value of 1 to a value of zero. Now, plug-in B has no dependencies and can be executed immediately" (Column 15, lines 21-35). The examiner further notes that **Young** teaches

**"adding the result set to the result set collection"** as "Returning to the example of FIG. 6A, plug-ins C and D can be executed immediately because they do not depend on other plug-ins, as indicated by the zero in the designations C:0 and D:0. Afterwards, as is the case each time a plug-in is executed, the dependency counts of counters 670A, . . . for all plug-ins that depend on the executed plug-ins are decremented by decrementer 672. Accordingly, as shown in FIG. 6C, when plug-in C is executed, the counter for plug-in B (which depends on plug-in C) will be decremented from a value of 2 to a value of 1. Plug-in B now has only one dependency, plug-in D. As shown in FIG. 6D, when plug-in D is executed, the counter for plug-in B will again be decremented, this time from a value of 1 to a value of zero. Now, plug-in B has no dependencies and can be executed immediately" (Column 15, lines 21-35). The examiner further notes that

**Young** teaches **"repeating steps (A)-(D) until all the specified functional modules have been executed"** as "Each plug-in 662, 664, or 666 includes a dependency counter 670A-C (or, alternatively, is associated with a counter stored in memory). Prior to execution, the execution management framework 652 causes the counter for each plug-in to be loaded with a count that indicates the number of plug-ins on which it depends. As a plug-in 662, 664, or 666 completes execution, it notifies the execution management framework 652. A count conditioning mechanism, e.g., the illustrated decrementer 672 in the framework 652, causes the counter 670A-C of any plug-ins dependent on the then executed plug-in to be decremented by a single count. This continues iteratively upon each execution until a predetermined threshold is reached, e.g., a count of zero, for a particular plug-in. (Alternatively, an incrementer could be

Art Unit: 2168

used, and incremented to a predetermined threshold.) When the threshold is reached for the counter 670A-C of a particular plug-in 662, 664, 666, that plug-in can be executed because its dependencies have been resolved" (Column 15, lines 3-20).

Regarding claim 12, **Young** further teaches a method comprising:

A) wherein the result set collection comprises results received in response to issuing a query (Column 14, lines 55-60)

The examiner notes that **Young** teaches "**wherein at least one of the functional modules is a plug-in component of the application**" as "Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed"(Column 14, lines 55-60).

Regarding claim 13, **Young** further teaches a method comprising:

A) wherein the interface is utilized to specify a plurality of functional modules by specifying a multi-analysis functional module (Column 3, lines 32-48, lines 59-63).

The examiner notes that **Young** teaches "**wherein the interface is utilized to specify a plurality of functional modules by specifying a multi-analysis functional module**" as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the

Art Unit: 2168

processing modules so as to take into account dependencies between them” (Column 3, lines 32-48) and “The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed” (Column 13, lines 59-67-Column 14, lines 1-6).

Regarding claim 14, **Young** further teaches a method comprising:

A) wherein obtaining the set of one or more parameters required for invoking the specified functional modules comprises retrieving information from a configuration file relating the multi-analysis functional module to the specified functional modules (Column 3, lines 32-48, Column 13, lines 59-67-Column 14, lines 1-6, Figure 6F)

The examiner notes that **Young** teaches “**wherein obtaining the set of one or more parameters required for invoking the specified functional modules comprises retrieving information from a configuration file relating the multi-analysis functional module to the specified**” as “the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in

Art Unit: 2168

the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6).

Regarding claim 15, **Young** teaches a computer-readable medium comprising:

- A) providing an interface for specifying a plurality of functional modules (Column 2, lines 58-67-Column 3, lines 1-4);
- B) providing a configuration file containing information regarding invocation of the functional modules (Column 2, lines 52-57, Column 9, lines 12-16, lines 54-63); and
- C) invoking the plurality of functional modules in a manner determined according to information retrieved from the configuration file (Column 9, lines 13-33).

The examiner notes that **Young** teaches "**providing an interface for specifying a plurality of functional modules**" as "The processing modules, referred to herein also as "plug-ins", can operate under the control of an execution management framework. The plug-ins can be viewed as plugging into and out of the framework, and as modular, reusable computational pieces or building blocks, which come together to perform the computation under the direction of the framework and pursuant to a configuration file. The plug-ins can be added, removed, and/or replaced for modifying

Art Unit: 2168

the calculation performed by the system in generating output data. For use, a user or operator devices the computation, e.g., calculation formula, required for a particular VAS, and uses the configuration manager's user interface to select and order the plug-ins to carry out that formula" (Column 2, lines 58-67-Column 3, lines 1-4). The examiner further notes that **Young** teaches "**providing a configuration file containing information regarding invocation of the functional modules**" as "Each processing module performs a specific sub-part of a computation on the metered information, and the configuration manager generates a configuration file for specifying the order of operation, computation sub-part, and other operational parameters for the individual processing modules" (Column 2, lines 52-57) and "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24). The examiner further notes that **Young** teaches "**invoking the plurality of functional modules in a manner determined according to information retrieved from the configuration file**" as "As illustrated, those of the plug-ins nos. 1-8 that can be processed in parallel are shown in a vertical stack in the drawing (as in the case, e.g., of plug-ins 1, 2 and 3; or plug-ins 6 and 7; or plug-ins 8 and 9). Moreover, those of the plug-ins nos. 1-8 that are dependent on, and therefore need to be processed after, other plug-ins are shown to the right of the other plug-ins on which they depend (as in the case, e.g., of plug-in 5 dependent on plug-in 4 and thus shown in the drawing, to the right of plug-in 4)" (Column 9, lines 24-33).

Art Unit: 2168

Regarding claim 16, **Young** further teaches a computer readable medium comprising:

A) wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the plurality of functional modules (Column 3, lines 32-48, lines 59-63); and

B) the configuration file contains information relating the plurality of functional modules to the multi-analysis functional modules (Column 3, lines 32-48, Column 13, lines 59-67-Column 14, lines 1-6, Figure 6F)

The examiner notes that **Young** teaches “**wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the plurality of functional modules**” as “the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them” (Column 3, lines 32-48) and “The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the

infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6). The examiner further notes that **Young** teaches **"the configuration file contains information relating the plurality of functional modules to the multi-analysis functional modules"** as "the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them" (Column 3, lines 32-48) and "The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed" (Column 13, lines 59-67-Column 14, lines 1-6).

Regarding claim 17, **Young** further teaches a computer readable medium comprising:

A) wherein the configuration file contains an explicit sequence in which the functional modules should be executed (Column 2, lines 52-57, Column 3, lines 32-48, Column 9, lines 12-16, lines 54-63)

The examiner notes that **Young** teaches “**wherein the interface is a graphic user interface utilized by users to specify functional modules**” as “the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them” (Column 3, lines 32-48).

Regarding claim 18, **Young** further teaches a computer readable medium comprising:

A) wherein the configuration file contains information indicating one or more parameters required for invoking each of the functional modules (Column 9, lines 12-16, lines 54-63, Column 14, lines 46-60); and

B) invoking the plurality of functional modules comprises invoking only those functional modules whose one or more required parameters are available (Column 9, lines 12-16, lines 54-63, Column 14, lines 46-60).

The examiner notes that **Young** teaches “**wherein the configuration file contains information indicating one or more parameters required for invoking**



Art Unit: 2168

**each of the functional modules”** as “A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins” (Column 9, lines 12-24) and “The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed” (Column 14, lines 46-60). The examiner further notes that **Young** teaches “**invoking the plurality of functional modules comprises invoking only those functional modules whose one or more required parameters are available**” as “A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in

Art Unit: 2168

sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

Regarding claim 19, **Young** further teaches a computer readable medium comprising:

A) wherein the configuration file is in an extensible markup language (XML) format (Column 10, lines 4-7)

The examiner notes that **Young** teaches "**wherein the interface is a graphic user interface utilized by users to specify functional modules**" as "The configuration manager 150 generates a configuration file for each stage of the pipeline 518, preferably specifying configuration data in XML format" (Column 10, lines 4-7).

Regarding claim 20, **Young** teaches a system comprising:

A) a plurality of functional modules (Column 2, lines 58-64)

B) providing a configuration file containing information regarding execution of the functional modules (Column 2, lines 52-57, Column 9, lines 12-16, lines 54-63); and

C) an application from which the functional modules are accessible (Column 2, lines 58-67-Column 3, lines 1-4);

D) wherein the application is configured to provide an interface for specifying a set of functional modules (Column 2, lines 58-67-Column 3, lines 1-4); and

E) execute the functional modules in a manner determined according to information retrieved from the configuration file (Column 9, lines 13-33).

The examiner notes that **Young** teaches **“a plurality of functional modules”** as “The processing modules, referred to herein also as “plug-ins”, can operate under the control of an execution management framework. The plug-ins can be viewed as plugging into and out of the framework, and as modular, reusable computational pieces or building blocks, which come together to perform the computation under the direction of the framework and pursuant to a configuration file” (Column 2, lines 58-64). The examiner further notes that **Young** teaches **“providing a configuration file containing information regarding execution of the functional modules”** as “Each processing module performs a specific sub-part of a computation on the metered information, and the configuration manager generates a configuration file for specifying the order of operation, computation sub-part, and other operational parameters for the individual processing modules” (Column 2, lines 52-57) and “A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins” (Column 9, lines 12-24). The examiner further notes that **Young** teaches **“an application from which the functional modules are accessible”** as “The processing modules, referred to herein also as “plug-ins”, can operate under the control of an execution management framework. The plug-ins can be viewed as plugging into and out of the framework, and as modular, reusable computational pieces or building blocks, which come together to perform the computation under the direction of the framework and pursuant to a configuration file. The plug-ins can be added, removed, and/or replaced for modifying the calculation

Art Unit: 2168

performed by the system in generating output data. For use, a user or operator devices the computation, e.g., calculation formula, required for a particular VAS, and uses the configuration manager's user interface to select and order the plug-ins to carry out that formula" (Column 2, lines 58-67-Column 3, lines 1-4). The examiner further notes that **Young teaches "wherein the application is configured to provide an interface for specifying a set of functional modules"** as "The processing modules, referred to herein also as "plug-ins", can operate under the control of an execution management framework. The plug-ins can be viewed as plugging into and out of the framework, and as modular, reusable computational pieces or building blocks, which come together to perform the computation under the direction of the framework and pursuant to a configuration file. The plug-ins can be added, removed, and/or replaced for modifying the calculation performed by the system in generating output data. For use, a user or operator devices the computation, e.g., calculation formula, required for a particular VAS, and uses the configuration manager's user interface to select and order the plug-ins to carry out that formula" (Column 2, lines 58-67-Column 3, lines 1-4). The examiner further notes that **Young teaches "execute the functional modules in a manner determined according to information retrieved from the configuration file"** as "As illustrated, those of the plug-ins nos. 1-8 that can be processed in parallel are shown in a vertical stack in the drawing (as in the case, e.g., of plug-ins 1, 2 and 3; or plug-ins 6 and 7; or plug-ins 8 and 9). Moreover, those of the plug-ins nos. 1-8 that are dependent on, and therefore need to be processed after, other plug-ins are shown to the right of the other plug-ins on which they depend (as in the case, e.g., of plug-in 5 dependent on plug-in 4 and thus shown in the drawing, to the right of plug-in 4)" (Column 9, lines 24-33).

Regarding claim 22, **Young** further teaches a system comprising:

A) wherein at least one of the functional modules is a plug-in component of the application (Column 2, lines 58-64)

The examiner notes that **Young** teaches "**wherein at least one of the functional modules is a plug-in component of the application**" as "The processing

Art Unit: 2168

modules, referred to herein also as "plug-ins", can operate under the control of an execution management framework. The plug-ins can be viewed as plugging into and out of the framework, and as modular, reusable computational pieces or building blocks, which come together to perform the computation under the direction of the framework and pursuant to a configuration file" (Column 2, lines 58-64).

Regarding claim 23, **Young** further teaches a system comprising:

A) wherein at least one of the plurality of functional modules is an external application (Column 2, lines 48-67).

The examiner notes that **Young** teaches "**wherein at least one of the plurality of functional modules is an external application**" as "The invention resides in a metering and processing system for processing metered information, which incorporates configurable processing modules and a configuration manager. The system can be readily and flexibly configured, responsive to operator directions, to process information to meet the needs of data consumers, such as NSPs and ISPs. Each processing module performs a specific sub-part of a computation on the metered information, and the configuration manager generates a configuration file for specifying the order of operation, computation sub-part, and other operational parameters for the individual processing modules" (Column 2, lines 46-57) and "The processing modules, referred to herein also as "plug-ins", can operate under the control of an execution management framework. The plug-ins can be viewed as plugging into and out of the framework, and as modular, reusable computational pieces or building blocks, which come together to perform the computation under the direction of the framework and pursuant to a configuration file" (Column 2, lines 58-64).

Regarding claim 24, **Young** further teaches a system comprising:

A) wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the specified set of functional modules (Column 3, lines 32-48, lines 59-63); and

Art Unit: 2168

B) the configuration file contains information relating the specified set of functional modules to the multi-analysis functional modules (Column 3, lines 32-48, Column 13, lines 59-67-Column 14, lines 1-6, Figure 6F)

The examiner notes that **Young** teaches “**wherein the interface is utilized to specify a single multi-analysis functional module used to invoke the specified set of functional modules**” as “the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them” (Column 3, lines 32-48) and “The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed” (Column 13, lines 59-67-Column 14, lines 1-6). The examiner further notes that **Young** teaches “**the configuration file contains information relating the specified set of functional**

**modules to the multi-analysis functional modules**” as “the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them” (Column 3, lines 32-48) and “The pipeline operator can combine plug-ins into large chunks of functionality, called stages. Each stage is made up of several pipeline plug-ins. The pipeline infrastructure calls each plug-in in a stage in the correct order, managing the dependency and load balancing between the plug-ins. Communication between stages is based on a queuing architecture. Each stage manages a queue of messages, e.g., session objects. The pipeline infrastructure pulls the session objects off the queues when appropriate, and then calls the plug-ins in the right dependency order to process the session objects. After all the plug-ins have been processed in the stage, the infrastructure sends the session objects to the queues of additional stages. Any fork plug-ins redirect sessions to stages different from the ordinary sequence of stages in which the plug-ins would be processed” (Column 13, lines 59-67-Column 14, lines 1-6).

Regarding claim 25, **Young** further teaches a system comprising:

A) wherein the configuration file contains an explicit sequence in which the functional modules should be executed (Column 2, lines 52-57, Column 3, lines 32-48, Column 9, lines 12-16, lines 54-63)

The examiner notes that **Young** teaches “**wherein the interface is a graphic user interface utilized by users to specify functional modules**” as “the configuration manager can further include a dependency data table for maintaining data regarding computational dependencies; and an order determining mechanism, responsive to the dependency data, for tracking computational dependencies for the plug-ins and insuring the proper order of plug-in operation. The order determining mechanism can include a dependency counter associated with each plug-in for determining an operational sequence position of the associated plug-in, and means for conditioning the counter (e.g., decrementing or incrementing the count) in response to operation of another of the plug-ins on which the associated plug-in depends, so that the associated plug-in will commence operation in the proper order when the counter reaches a predetermined count or value. Thus, in conclusion, a mechanism is provided for tracking and enforcing the ordering of data processing by the processing modules so as to take into account dependencies between them” (Column 3, lines 32-48).

Regarding claim 26, **Young** further teaches a system comprising:

- A) wherein the configuration file contains information indicating one or more parameters required for invoking each of the specified set of functional modules (Column 9, lines 12-16, lines 54-63, Column 14, lines 46-60); and
- B) invoking the specified set of functional modules comprises invoking only those functional modules whose one or more required parameters are available (Column 9, lines 12-16, lines 54-63, Column 14, lines 46-60).

The examiner notes that **Young** teaches “**wherein the configuration file contains information indicating one or more parameters required for invoking each of the specified set of functional modules**” as “A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to



Art Unit: 2168

determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60). The examiner further notes that **Young** teaches **"invoking the specified set of functional modules comprises invoking only those functional modules whose one or more required parameters are available"** as "A stage configuration module 416 receives configuration files 418 from the configuration manager 150, which define stage operations as well as operation of the plug-ins nos. 1-8 of the corresponding process space 404 and their processing interdependencies. The stage configuration module passes the plug-in configuration information to an execution management framework 425. The execution management framework 425 uses this information to determine which of the plug-ins nos. 1-8 can be processed in parallel (and during the same clock cycles per clock 420) and which of the plug-ins nos. 1-8 need to be processed in sequence after other plug-ins because they depend on a final or intermediary result from the other plug-ins" (Column 9, lines 12-24) and "The configuration file holds information associated with each node in the graph 600 as needed to execute the plug-in, including information specifying the number of other plug-ins on which each plug-in depends. The stage uses that information in executing the plug-ins in the correct order. At any given time, it might be possible to execute more

Art Unit: 2168

than one plug-in: if two plug-ins do not depend on each other, the two plug-ins can be executed in any order, or can be executed simultaneously (i.e., in parallel during the same clock cycle) by multiple threads. Therefore, for instance, if a plug-in will take a long time to execute because it must wait on the results of a database query by another plug-in, the pipeline infrastructure can execute other plug-ins in the mean time. At any time, any plug-in that has zero dependencies can be executed" (Column 14, lines 46-60).

***Claim Rejections - 35 USC § 103***

8. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

This application currently names joint inventors. In considering patentability of the claims under 35 U.S.C. 103(a), the examiner presumes that the subject matter of the various claims was commonly owned at the time any inventions covered therein were made absent any evidence to the contrary. Applicant is advised of the obligation under 37 CFR 1.56 to point out the inventor and invention dates of each claim that was not commonly owned at the time a later invention was made in order for the examiner to consider the applicability of 35 U.S.C. 103(c) and potential 35 U.S.C. 102(e), (f) or (g) prior art under 35 U.S.C. 103(a).

9. Claim 21 is rejected under 35 U.S.C. 103(a) as being unpatentable over **Young** (U.S. Patent 6,560,606) as applied to claims 1-20, and 22-26 and in view of **Manolis et al.** (U.S. Patent 6,583,799).

10. Regarding claim 21, **Young** does not explicitly teach a method comprising:  
A) wherein the application is a query building application.

**Manolis**, however, teaches "**wherein the application is a query building application**" as "Referring now to FIG. 3, a process 300 for operating on an image file is shown. First, a viewer is launched (step 301). The viewer could be a browser such as

Art Unit: 2168

Internet Explorer, available from Microsoft Corporation of Redmond Washington, or Netscape, available from Netscape, Inc. of Mountain View, Calif. (now America Online). The browser loads a set of embedded tags such as HTML tags that instruct the browser to instantiate or load a software module such as a plug-in. The plug-in is a software program that extends the capability of the browser in a specific way - providing the ability to receive images dragged over an area defined by the plug-in. The browser does not need to run an external application in order to interpret the data, and the result can be embedded as part of the Web page" (Column 4, lines 62-67-Column 5, lines 1-7).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of the cited references because teaching **Manolis's** would have allowed **Young's** to provide a method for presenting a user with a single unified user interface where the Web document is a container for many different media types, as noted by **Manolis** (Column 5, lines 7-9).

### ***Conclusion***

11. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

U.S. PGPUB 2004/0073782 issued to **Price et al.** on 15 April 2004. The subject matter disclosed therein is pertinent to that of claims 1-26 (e.g., methods to manage/use plug-ins).

U.S. Patent 6,782,531 issued to **Young** on 24 August 2004. The subject matter disclosed therein is pertinent to that of claims 1-26 (e.g., methods to manage/use plug-ins).

U.S. PGPUB 2004/0205695 issued to **Fletcher** on 14 August 2004. The subject matter disclosed therein is pertinent to that of claims 1-26 (e.g., methods to manage/use plug-ins).


U.S. PGPUB 2004/0010791 issued to **Jain et al.** on 15 January 2004. The subject matter disclosed therein is pertinent to that of claims 1-26 (e.g., methods to manage/use plug-ins).


**Contact Information**


12. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Mahesh Dwivedi whose telephone number is (571) 272-2731. The examiner can normally be reached on Monday to Friday 8:20 am – 4:40 pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tim Vo can be reached (571) 272-3642. The fax number for the organization where this application or proceeding is assigned is (571) 273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

  
February 26, 2007

Leslie Wong   
Primary Examiner

  
TIM VO  
SUPERVISORY PATENT EXAMINER  
TECHNOLOGY CENTER 2100

Mahesh Dwivedi  
Patent Examiner  
Art Unit 2168